AmarOk

Geste de savoir

p2p internals #2

22 septembre 2019

Table des matières

Lors du premier article de cette série, nous avons vu comment réaliser une application (*DEL*) transférant des fichiers d'un pair à un autre en réussissant à contourner le NAT via un serveur TURN. Cependant, cette technique est relativement peu efficace, dans bien des cas, le passage par un serveur TURN est inutile. En effet, il n'y a généralement pas de NAT à contourner, par exemple lors d'un transfert sur un même réseau ou encore entre deux pairs possédant des addresses IPv6. *Alice* se dit alors qu'elle peut améliorer *DEL* pour utiliser un serveur TURN seulement lorsqu'il n'est pas vraiment possible de faire autrement.

1. ICE, ou comment choisir le meilleur chemin

Il existe de nombreux chemins possibles entre deux pairs. En effet, un pair peut avoir de nombreuses addresses IPs. Disons par exemple l'addresse ip locale (en IPv4, en IPv6), l'addresse ip publique (en IPv4, en IPv6) et disons comme pour notre application, une addresse relayée par un serveur TURN (IPv6). Ce qui fait déjà 5 adresses possibles. Si le pair possède 5 adresses lui aussi, il y a déjà 25 chemins possibles ! De plus, la plupart des chemins ne fonctionneront pas (ipv4->ipv6, NAT, etc) et différents protocoles vont être mis en jeu (TURN, STUN, etc). Très vite, choisir le meilleur chemin devient relativement complexe. C'est cette complexité que gère ICE.

Actuellement ICE est majoritairement décrit dans 3 RFCs, (mais dépend de nombreuses autres RFCs que nous ne détaillerons pas dans cet article) :

— Le RFC 5245 déprécié aujourd'hui mais qui contient tout de même un petit morceau intéressant détaillé plus bas.
— Le RFC 6544 décrivant ICE sur TCP, la première ne décrivant que la méthode pour UDP.
— Le RFC 8445 étant la nouvelle norme.

2. Contournons le TURN (si possible)

Pour que *DEL* puisse utiliser le TURN uniquement en fallback et supporter ICE, *Alice* va devoir implémenter les 3 étapes que ICE demande.

2. Contournons le TURN (si possible)

2.0.1. Construire une liste de candidats.

La première étape d'ICE est de construire une liste de candidats et de les prioriser. Pour se faire, il va falloir trouver tous les couples (ip, port) que l'application utilise. Ainsi, imaginons que l'application écoute sur le port 1412 en local, correspondant au port 1214 sur le routeur (UPnP viendra dans un article futur), ou au port 4211 sur le TURN. On aura donc :

```
(local_v4, 1412) host
(local_v6, 1412) host
(public_v4, 1214) srflx
(public_v6, 1214) srflx
(turn_v6, 4211) srflx
```

Note : `host` et `srflx` (Server Reflexive) correspondent au type des candidats.

Chaque candidat est ensuite priorisé via la formule décrite ici

Enfin, le type de transport de candidat est ajouté (UDP, TCP passive, TCP active, TCP simultaneous open).

2.0.2. Testons les combinaisons

Lorsque les pairs se sont échangés leur listes respectives de candidats, ICE va alors commencer à négocier le meilleur pair. Tout d'abord en construisant une liste des checks à faire (processus décrit ici). Ceci permet notamment d'éviter d'avoir à tester les 25 liens mentionnés dans l'exemple plus haut.

Puis, cette checklist est parcourue, en faisant les checks de connectivité dans l'ordre. Une énorme différence existe à ce point entre les RFC 5245 et 8445. En effet, la première parle d'une nomination aggressive qui permet de considérer tous les pairs comme un pair nominé (qui sera utilisé comme lien) possible. Ce type de nomination a tout de même pour avantage de supprimer un peu de latence, même si le pair nominé a possibilité de changer durant le processus.

2.0.3. Choisir le meilleur pair

Maintenant que nous connaissons quels chemins ont réussi. Il est enfin possible de nominer un candidat (par priorité). Ainsi, si Bob et Alice veulent de nouveau s'échanger un fichier, il est dorénavant peu probable qu'un serveur TURN soit nécessaire.

ICE est un énorme morceau et est relativement compliqué car il interagit avec de nombreux autres protocoles. Cependant, vous avez surement déjà vu des applications l'utilisant. Par exemple, lors des JZdS pour tous car Discord utilise ICE pour effectuer les connexions UDP entre les participants (et WebRTC en général).

De plus, même si le protocole est relativement lourd, il existe aussi des implémentations plus légères .

2. Contournons le TURN (si possible)

Si vous souhaitez aller un peu plus loin et ne souhaitez pas lire les RFCs en anglais, il est possible de lire les articles disponibles sur bortzmeyer.org :

— Le RFC 5245
— Le RFC 6544
— Le RFC 8445